Prepared		Document Name		
H. Jablonski, SW APEBU		TSC_internal_CAN_BMW_CCU.docx		
Checked	Approved	Date	Rev.	Project / Reference
		15.10.2020	00	BMW CCU

Technical safety concept ECU-internal communication bus (based on CAN2.0 standard) in BEV and PHEV CCU ECUs.

Prepared		Document Name		
H. Jablonski, SW APEBU		TSC_internal_CAN_BMW_CCU.docx		
Checked	Approved	Date	Rev.	Project / Reference
		15.10.2020	00	BMW CCU

1. Abstract

This document describes the safety mechanisms to be used on ECU-internal communication bus (based on CAN 2.0 standard) in BEV and PHEV CCU ECUs.

Prepared		Document Name		
H. Jablonski, SW APEBU		TSC_internal_CAN_BMW_CCU.docx		
Checked	Approved	Date	Rev.	Project / Reference
		15.10.2020	00	BMW CCU

2. Revision History

Rev.	Date	Author	Change Description
00	15.10.2020	H. Jablonski	Initial revision
01			
02			

Prepared		Document Name		
H. Jablonski, SW APEBU		TSC_internal_CAN_BMW_CCU.docx		
Checked	Approved	Date	Rev.	Project / Reference
		15.10.2020	00	BMW CCU

3. Content

Prepared		Document Name		
H. Jablonski, SW APEBU		TSC_internal_CAN_BMW_CCU.docx		
Checked	Approved	Date	Rev.	Project / Reference
		15.10.2020	00	BMW CCU

4. Safety mechanisms on ECU-internal communication bus

The ECU-internal communication bus based on CAN 2.0 uses 3 different safety mechanisms:

Mechanism	Description
Counter	4bit (explicitly sent) representing numbers from 0 to 15 incremented on every send request.
Timeout monitoring	Timeout is determined by evaluation of the counter and/or the message reception (timeout time 3 message cycle times)
CRC	CRC-8-CCITT polynomial: $x^8 + x^2 + x + 1$ implementation: lookup table (see chapter 5)

The mechanisms can detect the following faults or effects of faults:

Mechanism	Detected communication faults
Counter	Repetition, Loss, insertion, incorrect sequence, blocking
Timeout monitoring	Loss, delay, blocking
CRC	Corruption, Asymmetric information

4.1 Counter

On the sender side, for the first transmission request of a data element the counter shall be initialized with 0 and shall be incremented by 1 for every subsequent send request (from sender SW-C). When the counter reaches the value 15 (0xF), then it shall restart with 0 for the next send request.

4.2 CRC calculation

The ECU-internal communication bus uses CRC8-CCITT calculated using all 7 application data bytes including the incremented counter value for the next transmit request. The CRC calculation is to be implemented using lookup table and “Crc_CalculateCRC8” function as described in chapter 5.

4.3 Timeout detection

The timeout detection is to be implemented as CAN message timeout for CAN messages with fixed cycle time. The timeout time is specified to 3 cycle times.

4.4 Error reaction / fault tolerance

An error reaction / DTC shall be triggered in case:

- calculated CRC does not match the received CRC (fault tolerance 3 wrong CRCs)
- counter is not incremented (fault tolerance 3 skipped / repeated counter values)
- CAN message timeout (timeout time = 3 CAN message cycle times)

4.5 Error reaction on affected µController / DSP

The error reaction on the affected µController / DSPs is currently not scope of this document.

5. CRC implementation

```
static const uint8 CrcTable08[256u] = {
  0x00u, 0x07u, 0x0eu, 0x09u, 0x1cu, 0x1bu, 0x12u, 0x15u, /*   7 */
  0x38u, 0x3fu, 0x36u, 0x31u, 0x24u, 0x23u, 0x2au, 0x2du, /*  15 */
  0x70u, 0x77u, 0x7eu, 0x79u, 0x6cu, 0x6bu, 0x62u, 0x65u, /*  23 */
  0x48u, 0x4fu, 0x46u, 0x41u, 0x54u, 0x53u, 0x5au, 0x5du, /*  31 */
  0xe0u, 0xe7u, 0xeeu, 0xe9u, 0xfcu, 0xfbu, 0xf2u, 0xf5u, /*  39 */
  0xd8u, 0xdfu, 0xd6u, 0xd1u, 0xc4u, 0xc3u, 0xcau, 0xcdu, /*  47 */
  0x90u, 0x97u, 0x9eu, 0x99u, 0x8cu, 0x8bu, 0x82u, 0x85u, /*  55 */
  0xa8u, 0xafu, 0xa6u, 0xa1u, 0xb4u, 0xb3u, 0xbau, 0xbdu, /*  63 */
  0xc7u, 0xc0u, 0xc9u, 0xceu, 0xdbu, 0xdcu, 0xd5u, 0xd2u, /*  71 */
  0xffu, 0xf8u, 0xf1u, 0xf6u, 0xe3u, 0xe4u, 0xedu, 0xeau, /*  79 */
  0xb7u, 0xb0u, 0xb9u, 0xbeu, 0xabu, 0xacu, 0xa5u, 0xa2u, /*  87 */
  0x8fu, 0x88u, 0x81u, 0x86u, 0x93u, 0x94u, 0x9du, 0x9au, /*  95 */
  0x27u, 0x20u, 0x29u, 0x2eu, 0x3bu, 0x3cu, 0x35u, 0x32u, /* 103 */
  0x1fu, 0x18u, 0x11u, 0x16u, 0x03u, 0x04u, 0x0du, 0x0au, /* 111 */
  0x57u, 0x50u, 0x59u, 0x5eu, 0x4bu, 0x4cu, 0x45u, 0x42u, /* 119 */
  0x6fu, 0x68u, 0x61u, 0x66u, 0x73u, 0x74u, 0x7du, 0x7au, /* 127 */
  0x89u, 0x8eu, 0x87u, 0x80u, 0x95u, 0x92u, 0x9bu, 0x9cu, /* 135 */
  0xb1u, 0xb6u, 0xbfu, 0xb8u, 0xadu, 0xaau, 0xa3u, 0xa4u, /* 143 */
  0xf9u, 0xfeu, 0xf7u, 0xf0u, 0xe5u, 0xe2u, 0xebu, 0xecu, /* 151 */
  0xc1u, 0xc6u, 0xcfu, 0xc8u, 0xddu, 0xdau, 0xd3u, 0xd4u, /* 159 */
  0x69u, 0x6eu, 0x67u, 0x60u, 0x75u, 0x72u, 0x7bu, 0x7cu, /* 167 */
  0x51u, 0x56u, 0x5fu, 0x58u, 0x4du, 0x4au, 0x43u, 0x44u, /* 175 */
  0x19u, 0x1eu, 0x17u, 0x10u, 0x05u, 0x02u, 0x0bu, 0x0cu, /* 183 */
  0x21u, 0x26u, 0x2fu, 0x28u, 0x3du, 0x3au, 0x33u, 0x34u, /* 191 */
  0x4eu, 0x49u, 0x40u, 0x47u, 0x52u, 0x55u, 0x5cu, 0x5bu, /* 199 */
  0x76u, 0x71u, 0x78u, 0x7fu, 0x6au, 0x6du, 0x64u, 0x63u, /* 207 */
  0x3eu, 0x39u, 0x30u, 0x37u, 0x22u, 0x25u, 0x2cu, 0x2bu, /* 215 */
  0x06u, 0x01u, 0x08u, 0x0fu, 0x1au, 0x1du, 0x14u, 0x13u, /* 223 */
  0xaeu, 0xa9u, 0xa0u, 0xa7u, 0xb2u, 0xb5u, 0xbcu, 0xbbu, /* 231 */
  0x96u, 0x91u, 0x98u, 0x9fu, 0x8au, 0x8du, 0x84u, 0x83u, /* 239 */
  0xdeu, 0xd9u, 0xd0u, 0xd7u, 0xc2u, 0xc5u, 0xccu, 0xcbu, /* 247 */
  0xe6u, 0xe1u, 0xe8u, 0xefu, 0xfau, 0xfdu, 0xf4u, 0xf3u  /* 255 */
};

/*******************************************************************************
 * Global functions (public to other modules)
 *******************************************************************************/

/*******************************************************************************
* Function:        Crc_CalculateCRC8
*
* Parameters:      Uint8* cp, Uint8 length, Uint8 offset
* Returned value:  crc
*
* Description:     Calculate CRC8 checksum over the the data block and except
*                  the start block (first three bytes).
* Input:           ucCRC8_Table[]
* Output:          -
* Calling:
*******************************************************************************/
uint8_t Crc_CalculateCRC8(const uint16 *Crc_DataPtr, uint8_t Crc_Length){
  uint8 idx;
  uint8 ucCrcIndex  = 0;
  uint8 crc         = LOW_BYTE;
  uint8 data;

  for (idx = 0; Crc_Length > idx; idx++) {
    data = (uint8)Crc_DataPtr[idx];
    ucCrcIndex = ((crc ^ data) & LOW_BYTE);
    crc = (crc >> 8) ^ CrcTable08[ucCrcIndex];
  }
  crc ^= LOW_BYTE;
  return crc;
```